

Convert a Planning Load File to an Essbase Load File

There are a ton of reasons to convert a planning load file to an Essbase load file. Maybe you are migrating a file from one environment to another, or simple want to load the file faster, but there are reasons to use the Essbase format.

Oracle is working on an enhancement that should be released in the next month or two that will use a load rule to load data to the app using the Essbase load format, which means the logging will be much improved, it won't stop at the first failed line, and it will log all the errors, just like the Planning load format. That is great news for those of us that use the planning format purely for the logging.

Performance

Before I get into the script, I want to touch on the speed of this method. The file I used, based on a real situation, was over 89 million lines (yes, that is correct, million), and took over 5 hours to load as a Planning file. It had to be split into three files to be under the 2GB limit, but it successfully loaded. The file was received late in the morning and had to be loaded before the start of the day, so a 5 to 6-hour processing time was unacceptable. By the way, yes, the file was sorted appropriately based on the sparse and dense settings.

I was able to build a unix/linux script using awk to convert this file to an Essbase load format and it only took about 9 minutes to convert. The improved load time was pretty drastic. It finished in under 15 minutes.

For testing, it was great, and it was perfect to improve the processing until the source system could rebuild the export in

the Essbase format. Just to reiterate, I added less than 10 minutes to convert the file, and reduced the load time by 4.5 hours, so it was worth the effort.

The Catch

Before I continue, if you are unfamiliar as to why the two load formats, here is the difference. Essbase loads the data directly to Essbase. The Planning load will bounce the file off the Planning repository to convert any smart list string account to the appropriate number, which is what is stored in Essbase. This process creates a new file on the server, in an Essbase load format, with the numeric representation of each smart list account. If you have no smart list conversions, this entire process is done for no reason, which was the case in this situation. So, this isn't the answer in every situation.

The Script

Before I get into the script, if you know me you know I love my Mac. One of the reasons is because I have the performance of a Mac, I can obviously can run Windows whenever I want, and I have the ability to run Bash scripts through the terminal. I am not a Bash scripting expert, but it is extremely powerful for things like this, and I am learning more as I need to build out functionality.

If you are a Windows user, you can install and use Linux Bash scripting in Windows 10. You can read about it [here](#).

There are several languages that can be used, but I chose AWK, which is a domain-specific language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems.

First the script. Here is it. I put the awk on multiple

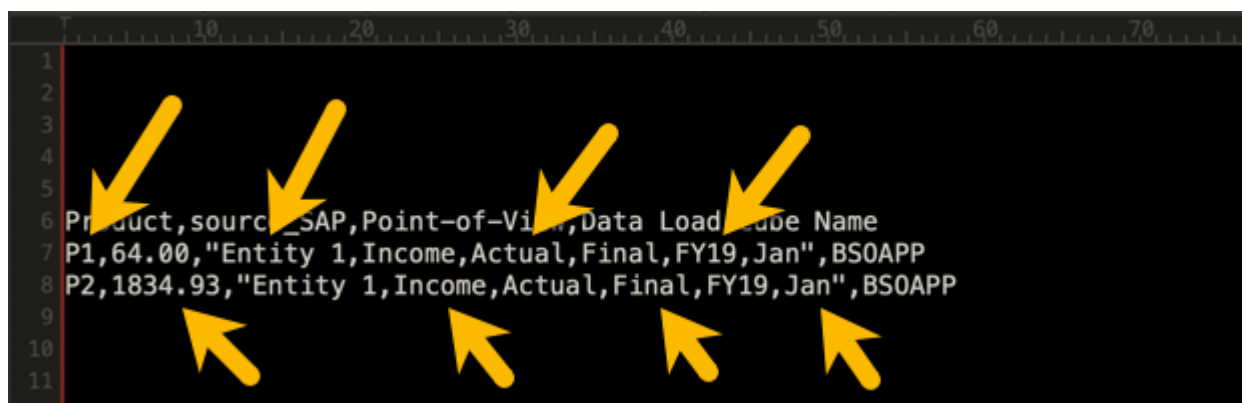
lines so it was a little more readable, but this is one command.

```
SOURCEFILE="Data.csv";  
LOADFILE="DataLoad.csv";  
HEADERMBR=$(head -1 $FILE | cut -d ',' -f2)
```

```
awk -v var="$HEADERMBR"  
  'BEGIN {FS=","; OFS="\t"}  
  NR>1  
  {gsub("/"/, "");  
   print "\"$1\"", "\"$3\"", "\"$4\"",  
         "\"$5\"", "\"$6\"", "\"$7\"",  
         "\"$8\"", "\"var\"", $2}'  
$SOURCEFILE > $LOADFILE;
```

There are a few things you will need to change to get this to work. Update the source file and the load file to reflect the file to be converted, and the file name of the converted file, respectfully. Inside the AWK script, I have 8 fields, 1 through 8. This represents the 8 columns in my Planning file, or the dimensions and the data. Your file might have a different count of dimensions. If your file has more or less delimited columns (ignore the POV field quotes and assume that each delimited field in that is an additional field), update the script as needed

In this example is a planning file example and each arrow represents a field. The print section of the awk command changes the column order to fit what the Essbase load format requires.



The screenshot shows a terminal window with a dark background and light text. The terminal output is as follows:

```
1  
2  
3  
4  
5  
6 Product,source_SAP,Point-of-View,Data Load,Cube Name  
7 P1,64.00,"Entity 1,Income,Actual,Final,FY19,Jan",BSOAPP  
8 P2,1834.93,"Entity 1,Income,Actual,Final,FY19,Jan",BSOAPP  
9  
10  
11
```

Yellow arrows point to the following fields in the CSV data:

- Arrow 1: Points to the first field (Product) in line 7.
- Arrow 2: Points to the second field (source_SAP) in line 7.
- Arrow 3: Points to the third field (Point-of-View) in line 7.
- Arrow 4: Points to the fourth field (Data Load) in line 7.
- Arrow 5: Points to the fifth field (Cube Name) in line 7.
- Arrow 6: Points to the first field (P1) in line 8.
- Arrow 7: Points to the second field (64.00) in line 8.
- Arrow 8: Points to the third field ("Entity 1,Income,Actual,Final,FY19,Jan") in line 8.
- Arrow 9: Points to the fourth field (BSOAPP) in line 8.

Breaking down AWK

This won't teach you everything there is to know about AWK, as I am still learning it, but it will explain the pieces used in this command so you can get started.

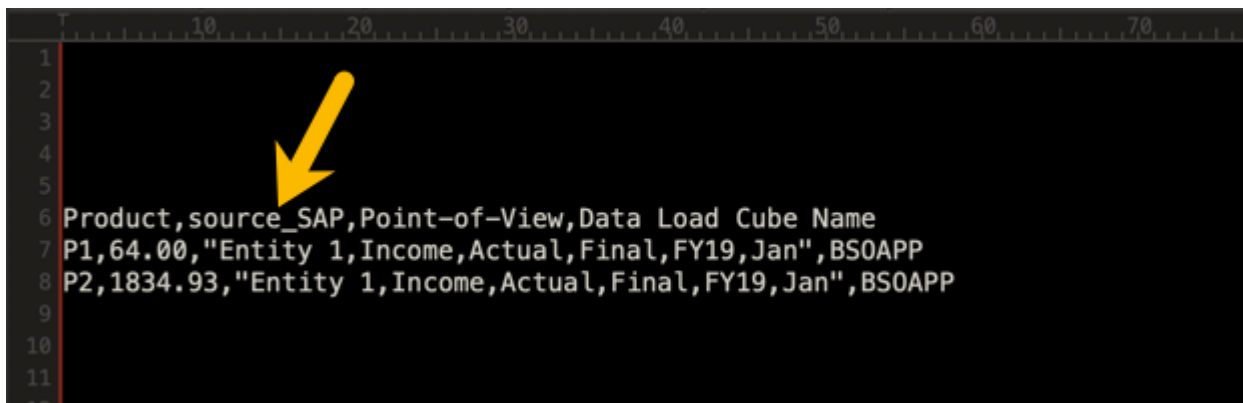
This piece is simply creating two variables, the source file and the converted file name, so there aren't multiple places to make these changes when the script needs updated.

```
SOURCEFILE="Data.csv";  
LOADFILE="DataLoad.csv";
```

The head command in Linux grabs specific lines, and -1 grabs the first line of the file. I pipe that with the cut command to grab the second field of the header line, which is the dimension member I need to add to every row. That gets stored in the HEADERMBR variable for later use.

```
HEADERMBR=$( head -1 $FILE | cut -d ',' -f2)
```

The example file above is repeated here. You can see that the second field is the member and HEADERMBR is set to source_SAP.



```
1  
2  
3  
4  
5  
6 Product,source_SAP,Point-of-View,Data Load Cube Name  
7 P1,64.00,"Entity 1,Income,Actual,Final,FY19,Jan",BSOAPP  
8 P2,1834.93,"Entity 1,Income,Actual,Final,FY19,Jan",BSOAPP  
9  
10  
11
```

Now the AWK command. Before I jump into it, the AWK looks like this.

```
awk 'script' filenames
```

And in the script above, the awk script has the following form.

```
/pattern/ { actions }
```

You can also think of pattern as special patterns BEGIN and END. Therefore, we can write an awk command like the following template.

```
awk '
    BEGIN { actions }
    /pattern/ { actions }
    /pattern/ { actions }
    .....
    END { actions }
' filenames
```

There are also a number of parameters that can be set.

This script starts with a variable. The -v allows me to create a variable. The first part of this command creates a variable named var and set it equal to the HEADERMBR value. I have to do this to use the variable in the script section.

```
-v var="$HEADERMBR"
```

The BEGIN identifies the delimiter as a comma and sets the output delimiter to a tab. FS and OFS are short for Field Separator and Outbound Field Separator.

```
'BEGIN {FS=","; OFS="\t"}
```

Since the file has a header file, and I don't want that in my Essbase load file, I only include the lines greater than 1, or skip the first line. NR>1 accomplishes that.

```
NR>1
```

Gsub allows me the ability to create substitutions. The source file has quotes around the POV field. AWK ignores the quotes, so it interprets the field with the start quote and the field with the end quote as a field with a quote in it. These need to be removed, so the gsub replaces a quote with a blank. The first parameter is a literal quote so it has to be "escaped" with a /.

```
gsub("/"/, "");
```

The next piece is rearranging the columns. I want to have the second column, or the column with the data, at the end. I have 8 columns, so I put them in the order of 1, skip 2, 3 through 8, then the variable that was created that has the dimension member in the header line, then 2(the data field). It looks a little clumsy because I append a quote before and after each field, which is required for the Essbase load format. But, this is just printing out the fields surrounded by quotes (except for field 2, the data field) and separated by columns.

```
print "\""$1\"\"", "\""$3\"\"", "\""$4\"\"", "\""$5\"\"",  
"\""$6\"\"", "\""$7\"\"", "\""$8\"\"", "\""var\"\"", $2
```

The last piece is identifying the file I want to do all this work to.

```
$SOURCEFILE
```

I want to send the results to a file, not the screen, and the > tells the command to send the results to a new file.

```
> $LOADFILE
```

The Result

The outcome is a file that is slightly larger due to the additional quotes and replicating the member from the header in every row, normalizing the file. It is converted to a tab delimited file rather than a comma delimited file. The header is removed. The app name is removed. And the columns are slightly different as the data column was moved to the end.

```
1 Product,source_SAP,Point-of-View,Data Load Cube Name
2 P1,64.00,"Entity 1,Income,Actual,Final,FY19,Jan",BSOAPP
3 P2,1834.93,"Entity 1,Income,Actual,Final,FY19,Jan",BSOAPP
4
5
```



```
1 "P1" "Entity 1" "Income" "Actual" "Final" "FY19" "Jan" "source_SAP" 64.00
2 "P2" "Entity 1" "Income" "Actual" "Final" "FY19" "Jan" "source_SAP" 1834.93
3
4
5
```

That's A Wrap

I am not ashamed to say this simple, basically one line script, took me forever to build and get to work. By forever, I don't mean days, but definitely hours. That is part of the learning process though, right? It was still quicker than waiting 6 hours for the file to load! So now you have basically a one line awk command that converts a Planning load file (or an export from Planning) to an Essbase load file and you can get home to have dinner.